OULU UNIVERSITY OF
APPLIED SCIENCES

Jari-Pekka Teurajärvi

JAVA LAUNCHER FOR SIMULATION TOOL

JAVA LAUNCHER FOR SIMULATION TOOL

Jari-Pekka Teurajärvi
Master's Thesis
Autumn 2013
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Jari-Pekka Teurajärvi
Title of Master's thesis: Java Launcher for Simulation Tool
Supervisor: Dr. Kari Laitinen
Term and year of completion: Autumn 2013 Number of pages: 42 + 21 appendices

---

This Master's thesis was commissioned by Nokia. Java MIDlets are applications that can be run in small devices such as mobile phones. In Nokia feature phones, MIDlets can be tested with Simulation tool.

Simulation tool users had a need to install Java MIDlets from the computer file system to the simulation environment file system. The purpose was to implement an easy graphical user interface for Java MIDlet file browsing, selecting and installing.

Simulation tool was designed as platform independent so it can be used with Windows and Linux OS. The file system of the Windows and Linux OS are different and the aim was to study the differences and take them into account when implementing the new feature to Simulation tool.

The final outcome of this work was a GUI application for installing the Java MIDlets from the computer file system to the simulation environment file system. The development proposal detected was to implement an automated start-up of the installed Java MIDlet to the Simulation tool.

Keywords:

Simulation tool, Java, MIDlet, file system

FOREWORD

This Master's Thesis is made for the Mobile Phones department of Nokia. Thesis materials will be used globally in Nokia Mobile Phones software development projects. The work was supervised by Dr. Kari Laitinen, Principal Lecturer of Oulu University of Applied Sciences and the supervisor from the company was Mr Saku Huttunen, Product Owner.

Oulu, Autumn 2013

Jari-Pekka Teurajärvi

CONTENTS

TERMS AND ABBREVIATIONS

API	An Application Programming Interface is a particular set of rules ('code') and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers. [9]
CD	The compact disc is an optical disc used to store digital data. [16]
Chipset	A chipset is a set of electronic components in an integrated circuit that manage the data flow between the processor, memory and peripherals. [12]
CLDC	The Connected Limited Device Configuration is a specification of a framework for Java ME applications describing the basic set of libraries and virtual-machine features that must be present in an implementation. [7]
drop-down list	A user interface control element which allows the user to choose one value from a list. [15]
DVD	A digital videodisk, a digital versatile disc is a an optical disc storage format. DVDs offer higher storage capacity than compact discs while having the same dimensions. [17]
EOF	End Of File
Ext	Extended file system is the first file system created specifically for the Linux. [20]
FAT	File Allocation Table is the name of the computer file system architecture and family of the industry standard file system utilizing it. [18]
file system	A technique for storing data in an organized and a human-readable form.

HAL	A hardware abstraction layer is an abstraction layer, implemented in software, between the physical hardware of a computer and the software that runs on that computer. [11]
HITL	Human-in-the-loop is defined as a model that requires human interaction. [27]
IOP	Interoperability is a property referring to the ability of diverse systems and organizations to work together (inter-operate). [10]
JAD	Java Application Descriptor files describe the MIDlets that are distributed as JAR files. [14]
JAR	Java ARchive is an archive file format typically used to aggregate many Java class files and associated metadata and resources (text, images and so on) into one file to distribute application software or libraries on the Java platform. [13]
Java ME	Java Platform, Micro Edition, is a Java platform designed for embedded systems. [6]
MIDlet	A MIDlet is an application that uses the Mobile Information Device Profile (MIDP) of the Connected Limited Device Configuration (CLDC) for the Java ME environment. [7]
MIDP	Mobile Information Device Profile is a specification published for the use of Java on embedded devices such as mobile phones and PDAs. [8]
NTFS	New Technology File System is a proprietary file system developed by Microsoft Corporation for its Windows operating systems [19]
OS	Operating System
Path	A general form of the name of a file or a directory that specifies a unique location in a file system. [22]
PDA	Personal digital assistant

UFS	A Unix file system is a file system used by Unix OS. [21]
UI	User Interface
XAML	Extensible Application Markup Language is a declarative XML-based language created by Microsoft that is used for initializing structured values and objects. [28]
XML	Extensible Markup Language is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. [29]

1 INTRODUCTION

Java based applications called MIDlets are an important piece of software in mobile devices like Nokia feature phones. MIDlets are independent applications running on top of the device OS. Most of the applications in case of Nokia feature phones are implemented using the Java. This Master's thesis is about a Java Launcher -feature implemented to a Simulation tool environment. The tool is a workstation simulation of a mobile device chipset hardware and a low level software, which enables an early stage prototype testing. Real device software is used from the UI to the chipset layer. The chipset layer has been replaced by the simulation. The application and the UI development can be carried out with the tool and fast prototyping enables a faster releasing of the software. Besides the software development, the tool enables testing and debugging. The tool has been designed as platform independent so it can be used with Windows and Linux OS.

In addition to the quicker software development cycle, the tool enables a cheaper development since additional testing or development devices are not necessary needed. Of course, the final testing should also be carried out in a target device environment. Software development is also quicker because the developer does not have to install the application to the device after every change but the testing can be done with the Simulation tool.

Java Launcher is needed by the software developers to enable an even faster development of their applications. With Java Launcher an application implemented by the software developer can easily be installed from the workstation file system to the Simulation tool file system. Usually the developers are willing to test the software after every new function or change in the software and that procedure is easier with Java Launcher.

2 SIMULATION

2.1 Simulation

Simulation is the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model is developed. The model represents the key characteristics or functions of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over a time. Simulation can be classified as a Physical or Interactive simulation. [25]

Simulator is a mechanism used to simulate the program in software itself. Simulation is not a replacement for emulation or real hardware.

2.1.1 Physical simulation

Physical simulation is a simulation where the real objects are replaced by a physical look-alike object. These physical objects are often chosen because they are smaller or cheaper than the actual object or system. [25]

2.1.2 Interactive simulation

Interactive simulation includes a human operation. Often these are also called Human-in-the-loop simulations and typically an HITL outcome is difficult to reproduce since every input can be different. [27]

The benefit of the Interactive simulation is that the outcome of the process can be changed every time by the user. Because of the flexibility, Interactive simulation is widely used for training skills that otherwise requires a big amount of money and time. The trained skills can be adopted in a real world action.

Examples of the Interactive simulations for training purposes are flight simulators, driving simulators, marine simulators and some video games. [27]

2.2 Emulation

Emulation refers to the ability of program or device to imitate another program or device. Emulation addresses the original hardware and software environment of the digital object, and recreates it in a current machine. The emulator allows the user to access any kind of program on a current platform, while the software runs as it did in its original environment. [26]

With the emulator the program is verified through hardware. An example of the emulator is a video game designed only for the video game console but which could be played on a PC using an emulator. [26]

2.3 Simulation versus emulation

It is quite often that simulation and emulation as a term are understood as synonymous. That is of course a wrong assumption. The easiest way to differentiate the terms is to understand that the simulation is done in software and the emulation is done in hardware. Figure 1 lists some of the differences between the simulation and emulation. The best practice is to use both simulation and emulation in a software development process.

Simulation	Emulation
used for the SW	used for the HW
used early in development	used later in development
insight into app behavior	insight into system behavior
deferred time	real-time

FIGURE 1. Differences between simulation and emulation.(Jari-Pekka Teurajärvi 2013)

2.4 Simulation in a development process

Simulation is used for the software development and not at all for the hardware development. In a software development process simulation allows developers to model the application even months before the actual hardware is available. With simulation the different configurations and variations of the software can be evaluated without the target device or prototype hardware. Simulation makes the whole software and device development process much faster and cheaper.

With the help of simulation the software can be implemented in a very efficient way and the performance of the code can be optimized in advance. Simulation makes it possible to write a unit test code for the software and debug the code. If the unit test code is written so that only the action HITL is needed to start the testing, the actual simulation test cases can be run identically over and over. With simulation the developer can also easily test many possible external events and interrupts.

2.5 Emulation in a development process

Emulation is used later in a software development process to be able to understand the application behavior in a real-time system with different hardware and software models. It is also possible to debug the application in a real-time system with external events and interrupts. Emulation enables verification of the simulator tested model by testing how the actual hardware performs when the application runs on it.

Emulation enables an accurate real-time hardware integration and system behavior testing. Emulation is not a replacement for a real hardware but it makes the integration possible even before the first hardware prototypes are available.

3 TASK DESCRIPTION AND TARGET

The purpose of this work was to implement a new feature to Simulation tool. The feature is supposed to ease the work of the software developers by providing a method to install a MIDlet from the workstation file system to a Simulation file system. As a workstation either Windows or Linux can be used.

3.1 Simulation tool

Simulation tool is a workstation simulation of a phone chipset hardware and a low level software, and it enables an early stage prototyping and testing of the applications in a workstation environment. The real device software is used but the chipset layer has been replaced by the simulation. For the software developers the tool provides an easy way to prototype an application or a user interface. Testing and debugging of the software can also be carried out by using Simulation tool.

With the help of Simulation tool the expenses of the software project can be reduced dramatically, because of an additional testing and prototyping, hardware costs are much lower than without Simulation tool. Also, the software can be released with a high quality to the market without compromising between the content or time on release. This is possible since usually the device hardware is not available or ready for testing when the application development is already proceeding at full speed.

Also later when the hardware is made available, the software is already mature enough to be released since the testing has been already done with Simulation tool. The maturity of the software means a smaller amount of errors and the testing effort can be focused on IOP and other directions that usually do not get enough attention.

Also, with Simulation tool you can easily create and test your new user interfaces and demonstrate them to the target group. With these demonstrations you can easily find out the weak point of your user interface and change your UI design.

3.1.1 Simulation tool architecture

As shown in Figure 2 the Simulation environment consists of three main modules: the tool, the simulations and the actual device software. There is the full device UI software and the core software running, but the hardware driver layers have been replaced with the simulations. The standard interfaces like HAL and Chipset API are supported, but in some cases it has not been possible to implement them. The most important simulations have been coded as fully functional systems using the Simulation tool internal library and user interface to offer some controlling possibilities for the developers. There are also some simulations that are implemented as non-functional to make it possible to build product software packages. It is possible to implement the non-functional simulations as functional simulations if needed. While the tool simulations are tailored for certain devices, the tool itself is an independent module which could be used as a tool for simulating any embedded device. The tool library offers a wide variety of real-time operating system functionalities, tracing systems and communication methods, which will make it possible to virtually anything with the tool user interface.

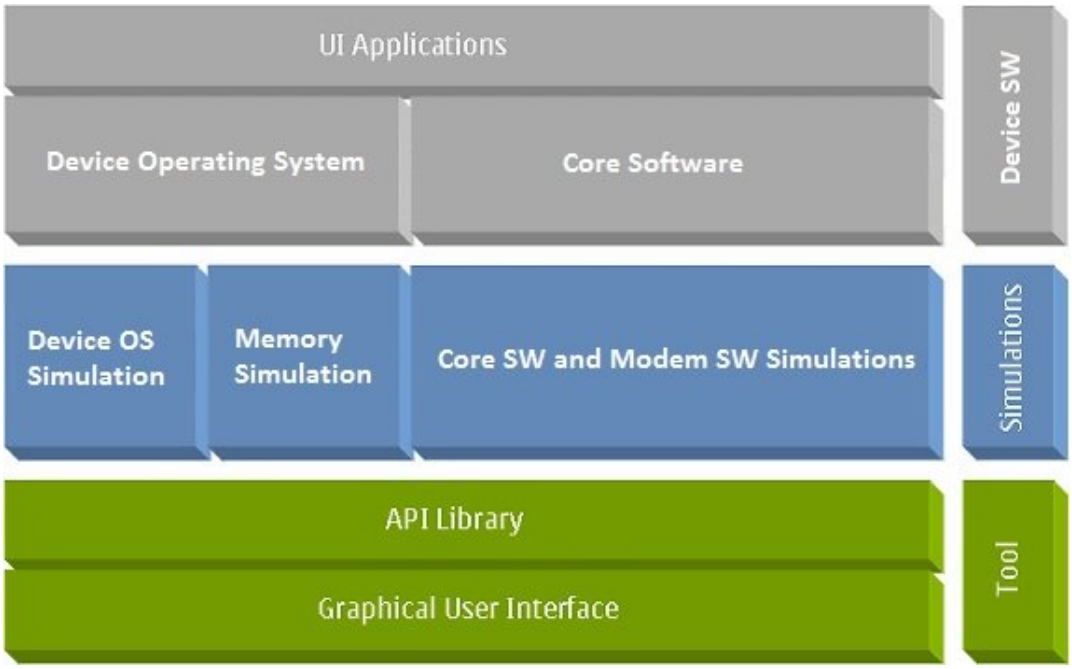

FIGURE 2. Simulation tool architecture (Nokia 2013).

The simulation environment has a multi-device support. The multi-device support enables a usage of several phones to communicate together, for example in a simulated conference call. It is also possible to run multiple phones in same workstations in such a manner that they cannot affect each other.

3.1.2 Simulation tool API library

Simulation tool has a library of different APIs to make the implementation easier and possible. There is an API for registering and unregistering entities for a common system functionality.

A communication subsystem consists of a channel API and Socket for sharing data between the simulation targets, GUI and 3rd party tools. The channel API is for handling all communication to the GUI client and communication between the different layers in simulation. The channel API and entity API can be considered as core building blocks in the simulation environment. Each channel has a memory area accessible from everywhere and therefore the difference

between the Windows OS and Linux OS memory management has been taken into account. Socket is a transport layer for both direct socket communication and channels.

An operating subsystem consists of several APIs for the abstraction of the host platform OS API calls. Scheduler is controlling the timing and internal threads. Internal timing is not real time but virtual time. The real time i.e. system time is handled by the host OS. There is a specific API for the time functionalities from a simulation code. A task API is providing a control for the simulation tasks. Tasks are usually controlled and synchronized by other simulation tasks. Task states are *suspended*, *ready* or *running*. A semaphore API provides methods for resource handling and inter-task signaling. Resources can be reserved, released or deleted. A mailbox API provides an asynchronous method for inter-task communication. Sent messages are queued into a mailbox which can be read by another task whenever the task is ready to read them.

A memory subsystem provides an API for memory allocation and freeing in the simulation code. If the simulated device shuts down, the memory allocated is automatically freed. With the memory API it is easy to test situations like out of memory.

A tracing subsystem API is used for tracing and logging the simulation execution and data flow. Traces can be directed to a log file for later usage.

An assert subsystem API enables to test conditions of program functionality. In case the condition is false an error message is displayed. For example if the system memory has run out of memory there will be assert.

A file subsystem API provides a POSIX-like API for file handling. The interface works equally on both supported host operating systems Linux and Windows.

A utilities subsystem includes generic helper functions to ease the implementation of the simulations.

A UI subsystem has an API for using the user interface components. A component can be for example a dialog.

With System API operations, specific operations to the host operating system can be run. One of the uses can be, for example, retrieving the system call stack.

3.2 The initial condition of Simulation tool

Installing MIDlets into Simulation tool before Java Launcher implementation was cumbrous. The users who were usually software developers had to manually copy the MIDlet files from the host OS file system to the Simulation tool file system. Users of course had to know the source location of the MIDlet files in a host OS but the more troubled part was to know the correct destination location folder to the MIDlet files into the simulation file system. The MIDlet files must have been copied into the specific destination folder to be able to launch the MIDlet in the simulation environment. The users had to know that both of the MIDlet files, the JAR file and the JAD file, must be copied into the same destination folder since they are dependent on each other. If there were old versions of the MIDlet files in a destination folder, they had to be deleted before copying new file versions. The deletion had to be done manually. When both of the MIDlet files were copied into the correct destination folder into the simulation environment, the user still had to browse the JAR file using Simulation tool and launch the MIDlet manually after finding the correct file.

3.3 The definition of the feature Java Launcher

Because of the problem described in the previous section, Simulation tool had to be equipped with a Java Launcher feature. The feature must provide an easy

way of copying MIDlet files from the host OS file system into the Simulation tool file system. The user has to know only the source location of the MIDlet files. Java Launcher will copy both the JAR and the JAD file into the destination folder. The destination folder is fixed and known by Java Launcher. Java Launcher will delete the old version of the MIDlet files if existing in Simulation tool destination folder. The user may choose if the MIDlet is launched automatically into Simulation tool.

3.4 The target of the feature Java Launcher

Target was to implement the feature incrementally so that the primary target was to enable the easy copying of the MIDlet files from the host OS file system into the simulation file system. The secondary target was to implement a user initiated automated launch of the MIDlet into Simulation tool right after successful copying.

Those two increments included a study if such features are possible to implement using the existing Simulation tool APIs and host OS APIs. It was also important to develop a user friendly graphical user interface for the usage of the Java Launcher. Java Launcher must be also traceable to be able to understand possible abnormal behavior.

It was mandatory Simulation tool requirement to have a help file for the feature. With a help file the usage of the feature Java Launcher can be explained detailed.

4 EXTENSIBLE APPLICATION MARKUP LANGUAGE

Extensible Application Markup Language (XAML) is a declarative XML-based language created by Microsoft that is used for initializing structured values and objects. It is available under Microsoft's Open Specification Promise. The acronym originally stood for Extensible Avalon Markup Language - Avalon being the nickname for Windows Presentation Foundation (WPF). [28]

XAML is used extensively in .NET Framework 3.0 & .NET Framework 4.0 technologies, particularly Windows Presentation Foundation (WPF), Silverlight, Windows Workflow Foundation (WF) and Windows Runtime XAML Framework. In WPF, XAML forms a user interface markup language to define UI elements, data binding, events, and other features. In WF, workflows can be defined using XAML. XAML can also be used in Silverlight applications, Windows Phone apps and Windows Store apps. [28]

4.1 Simulation tool GUI based on XAML

XAML provides a very easy way to create basic elements of the GUI. Some of the basic elements are Button, Menu, Checkbox, Combo box, Drop-down list, Radio button, Scrollbar, Text box and Status bar. With the usage of a XAML the elements drawn to the GUI are inherited from the used host OS. If MS Windows is used, the look and feel are the same as with any other MS Windows application. [3]

An example how to create some basic GUI elements using the XAML is illustrated in Figure 3. The Window Title named as “Testing” is created and 2 pieces of Button elements named as Button1 and Button2. The Button1 content “This is the 1st button” is displayed inside the Button element to the user. Similarly the Button2 content “This is the last button” is drawn.

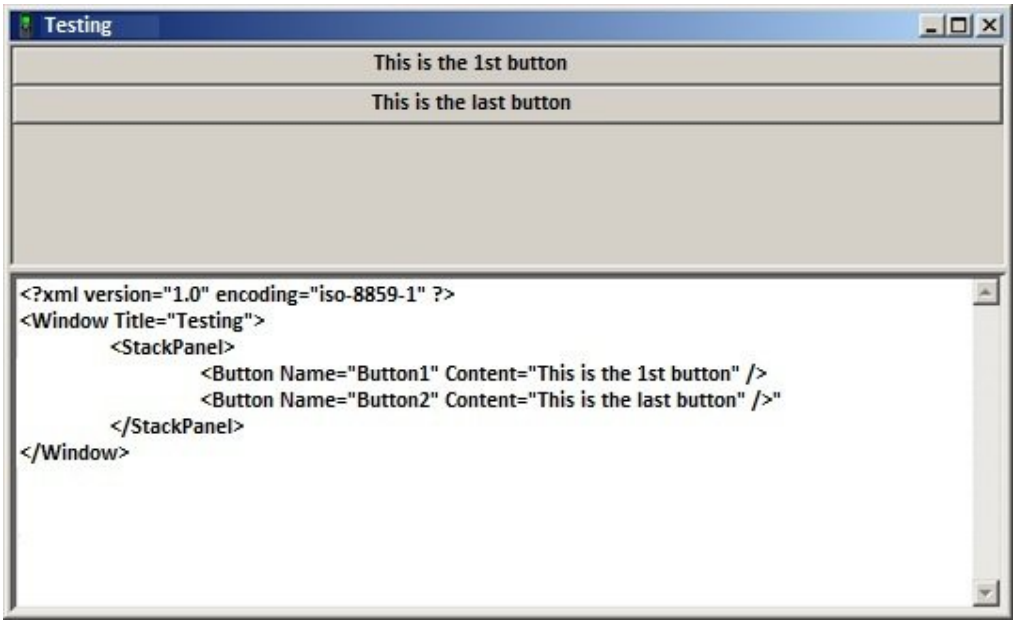

FIGURE 3. Example of a XAML created UI (Jari-Pekka Teurajärvi 2013).

Even the Simulation tool GUI implementation is based on XAML, the actual language used can be considered as an own markup language version. The UI components of Java Launcher were also implemented with XAML.

5 FILE SYSTEM

Simulation tool is designed as platform independent so it can be used with both Windows and Linux OS. The file systems of the Windows and Linux operating systems are different and therefore it was mandatory to study the differences and take them into account when implementing the new feature into Simulation tool.

5.1 File system in general

A file system is a technique for storing data in an organized and a human-readable form. The basic unit of a data file system is called a file. A file system is a very important component residing in most data storage devices like hard drives, CDs and DVDs. A file system helps the devices to maintain the physical location of the files. [2]

5.1.1 Windows file system

Windows mainly supports FAT and NTFS. FAT is the initial file system used in Windows. NTFS has a completely different data organization architecture. Basically, Microsoft developed NTFS to compete with UNIX, by replacing the much more simple FAT. A FAT partition can be easily converted to a NTFS partition without loosing data. Windows uses a drive letter to distinguish partitions. Traditionally, The C drive is the primary partition. The primary partition is used to install and boot Windows. The drive letter can be used for mapping network drives, too. [2][18][19]

5.1.2 Linux file system

Commonly used Linux file systems are the ext family and the XFS. The ext was the very first file system used in Linux OS. The ext is based on the UFS. The XFS is a journaling system with a high performance. In Linux, everything is a file. If something is not a file, then it is a process. Programs, audio and devices for example are considered as files. In Linux, there is no difference between a

file and a directory. A directory is simply a file containing names of a set of other files. [2]

5.1.3 Simulation tool file system

Simulation tool file system is a Windows like file system with the partitions and drive letters.

5.1.4 Differences between the Windows and Linux file system

In Linux, there is only a single hierarchal directory structure. Everything begins from the root directory which is represented by the symbol /, which then expands into sub-directories. Windows includes various partitions and then directories under those partitions. Linux places all the partitions underneath the root directory by mounting them in specific directories. In contrast, Windows uses the letter C as its root directory. [2]

The directory separator is also different between the Windows and Linux OS file system. The Windows uses the backslash (“\”) character and the Linux uses the slash character (“/”) as a delimiting character.

By those two differences the path format is different between the Windows and Linux OS file system. Example of the Windows path from the root of the file system is “C:\example\path.txt”. The similar path in a Linux file system is “/example/path.txt”.

5.2 File system APIs

Different operating systems have different file system APIs. It is mandatory to use the correct API to be able to enable the correct behavior of the software. The different file system APIs in practice provide the very same functionality. The basic operations provided by the file system APIs are write, read, position, open and close. The advanced service of the file system APIs are usually metadata management, file system maintenance, directory management, data

structure management, record management, sharing, restricting access and encryption.

5.2.1 Windows and Linux file API

Since Simulation tool is running in the Windows and Linux OS, the file handling can be implemented for the both operating systems using the very same standard library called “stdio.h”. That is because the library functions are already included in the Windows and Linux kernel and a programmer does not have to load the standard library separately. The “stdio.h” is a C programming language standard library. [1][22][23]

In case of Java Launcher the Java MIDlet copying from the computer file system to Simulation tool file system was needed. The file copying was implemented using the “stidio.h” standard library. The implemented procedure can be applied into any other similar file copying case. For the file access the “fopen” and “fclose” functions were used. For the direct input the “fread” function was used. For the file positioning the “fseek”, “ftell” and “rewind” functions were used. A more detailed description of the above mentioned functions are described below. [1][24][4]

The “fopen” in practice opens the file for reading and writing and enables the actual file access. The “fseek” and “ftell” can be used for determining the size of the file. With the “fseek” it is easy to find the EOF and after that the function “ftell” can be used to return the data position from the start of the file. The start position of the file is a zero point and the EOF is the end. The end position is telling the size of the file in bytes. After the file has been opened and the size of the file is known, it is possible to copy the content of the file by using the “fread” function. Before copying the file, it is mandatory to use the function “rewind” for settings the cursor position back to the beginning of the file. If the “rewind” is not done, the data will be read from the wrong position (EOF) and that will cause an error. After the previous procedures have been finished for the source file, the

file needs to be closed and that can be done using the “fclose” function. The file must be closed if no further action for the same file is needed. The programmer has to implement a possible error handling for the file handling procedures described above. The “stdio.h” standard library defines the possible error values when the library functions are used. For example, if the “fopen” is used for the file opening and the procedure fails, the function will get NULL as a return value. Figure 4 describes a file reading procedure using the stdio.h standard C library. [24][4]

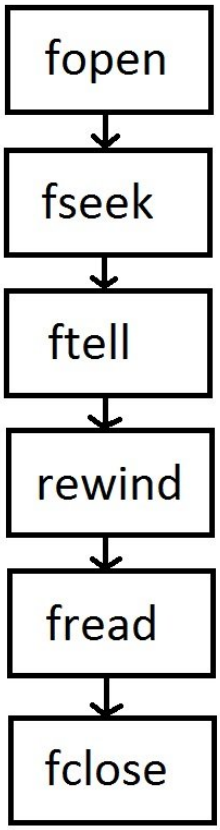

FIGURE 4. File reading procedure using the stdio.h standard C library (Jari-Pekka Teurajärvi 2013).

5.2.2 Simulation tool File API

Simulation tool provides a good API for the simulated device internal file handling. For Java Launcher purposes, the new empty file creation, data writing into

the new file and the file closing were needed to be able to complete the Java MIDlet copying from the computer file system to the Simulation tool file system. The Java MIDlet file data reading from the computer file system was implemented using the standard C library as described in Section 5.2.1. That enabled single one implementation for both of the operating systems, Windows and Linux. The data writing procedure was implemented using a Simulation tool File API and that of course is also valid for both of the operating systems with a single one implementation. The simulation tool File API was used because the Simulation tool internal file system was the target of the copied Java MIDlet.

The end user is able to select the Simulation tool file system drive partition for the Java MIDlet destination. That is determined by using the function “nose_file drivestat”.

If there exists an identically named Java MIDlet file in the Simulation tool file system, the file must be deleted before creating a new file named identically. That was done using the Simulation tool File API function “nose_file_delete”.

The “filesys_open” function was used for creating a new file. The actual function is similar to the standard C library function “fopen”. With the given parameter, the programmer can define that a new file creation will be done instead of opening an existing file.

After creating the new file, the Simulation tool File API function “filesys_write” was used to write the actual Java MIDlet file data to the new file. Finally, the function “filesys_close” was used to close the new file.

After the Java MIDlet copying, the Simulation tool file system should be refreshed so that the end user can see the latest status of the file system. The file system refresh was implemented using the function “nose_filesys_refresh”. The file writing procedure using the Simulation tool File API is illustrated in Figure 5.

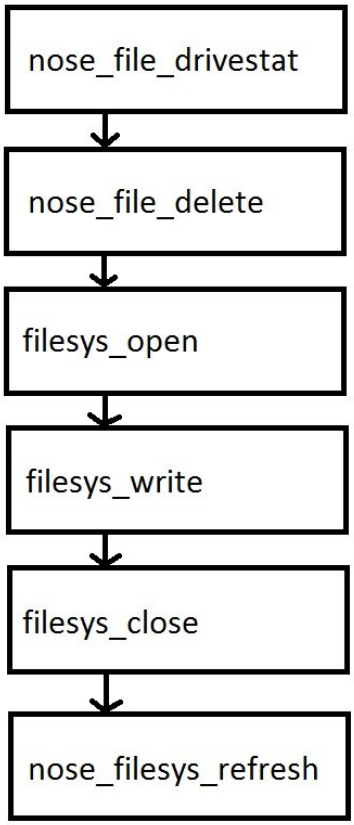

FIGURE 5. File writing procedure using Simulation tool File API (Jari-Pekka Teurajärvi 2013).

5.3 MIDlet launching in Simulation tool

It was also studied if the copied Java MIDlet can be launched right after copying. The Simulation tool API does not provide such a mechanism and the user must launch the MIDlet manually from the simulation GUI. As a future improvement, the Simulation tool API should be enhanced with the support of launching the Java MIDlets using the API.

6 JAVA LAUNCHER

Java Launcher is a new feature of Simulation tool. The purpose of Java Launcher is to provide a seamless mechanism for the end user to install Java MIDlets to Simulation tool, for example, for testing purposes. With the help of Java Launcher the end user experience with a virtual device used by Simulation tool is very close to the real world device user experience in case of installing the MIDlets. The complicated installation methods of the previous versions of Simulation tool are replaced with a modern and easy way of using a GUI tool.

6.1 Java Launcher flow chart

Figure 6 describes the functionality of Java Launcher.

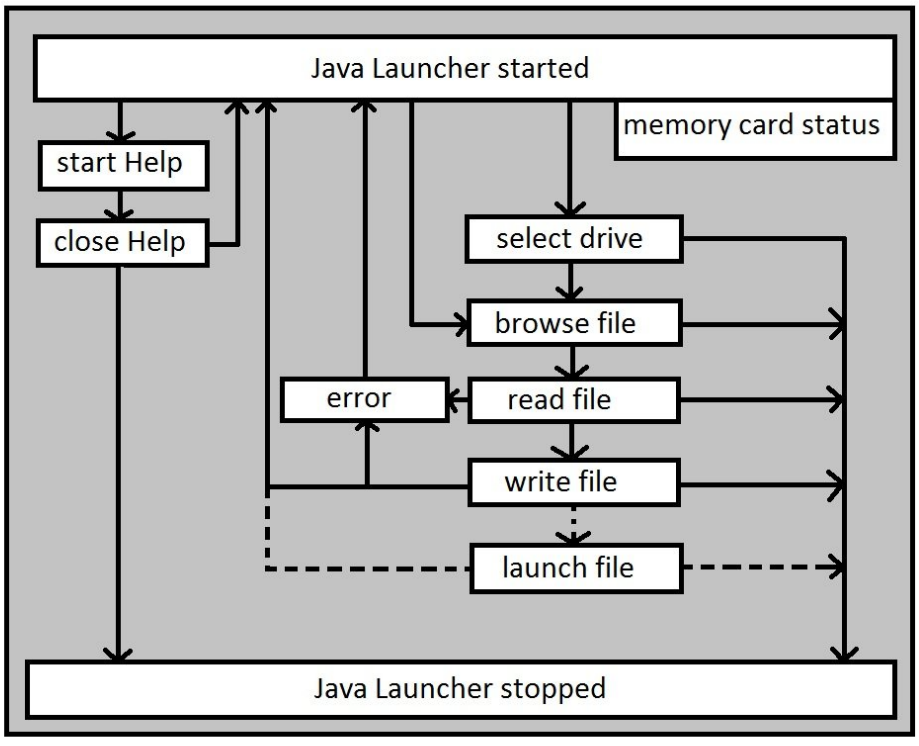

FIGURE 6. Java Launcher flow chart (Jari-Pekka Teurajärvi 2013).

Java Launcher is either started or stopped. The user may stop Java Launcher at any time. The file reading and writing is done seamlessly and it can be considered as file copying or file installing. The flow chart also describes an option for the file launching but it is not implemented into the first version of the feature since the current Simulation tool is not supporting the launch via API.

The error handling is centralized to make to the code reusable. The user may also start the Help section from the GUI and it is a separate part of the feature.

6.2 Starting Java Launcher

Java Launcher is a feature of Simulation tool and therefore Simulation tool itself must be started first. When Simulation tool is running, the end user can start Java Launcher from the Simulation tool Tools menu under the Java section. The Tools menu is described more detailed in Figure 7.

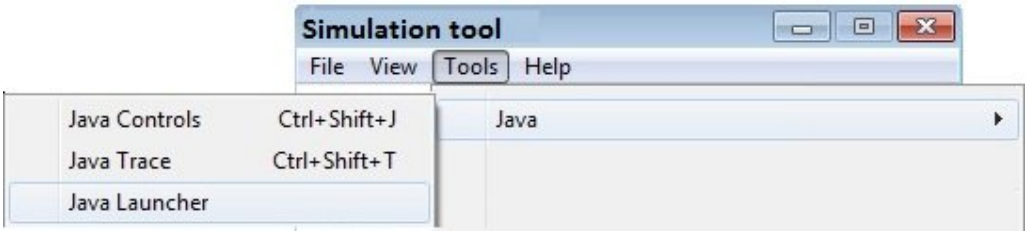

FIGURE 7. How to start Java Launcher (Jari-Pekka Teurajärvi 2013).

6.3 Java Launcher GUI

Java Launcher GUI is as simple as possible as seen in Figure 8. The GUI consists of the following elements: Help menu, Exit button, Memory card information note, Select memory area to be user for jar file, Browse button for browsing the jar file from the host OS file system and the Upload button for installing the MIDlet.

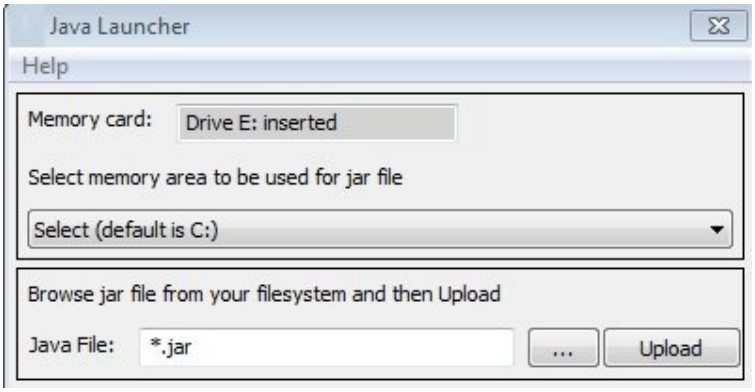

FIGURE 8. Java Launcher GUI (Jari-Pekka Teurajärvi).

6.3.1 Help menu

The Help menu is a mandatory part of Simulation tool to give more information about the feature if needed. Figure 9 displays the Help menu opening.

FIGURE 9. Help menu opening (Jari-Pekka Teurajärvi 2013).

The Help menu introduces the usage of Java Launcher tool and provides information for the FAQ. The help menu is implemented as a stand-alone pop-up window element. Figure 10 shows the Help menu content opened into the stand-alone window element.

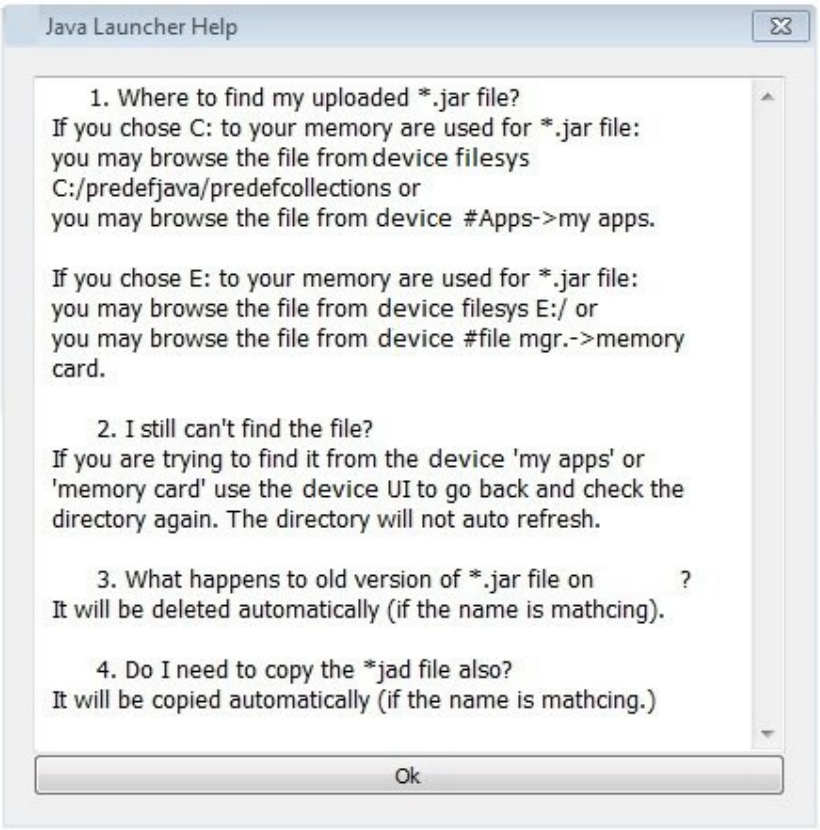

FIGURE 10. Help menu content (Jari-Pekka Teurajärvi 2013).

6.3.2 Exit button

The Exit button is a mandatory part of Simulation tool features like Java Launcher integrated into Simulation tool. The Exit button is familiar from the most of the Windows applications. In Figure 11 the Exit button is pointed with an arrow.

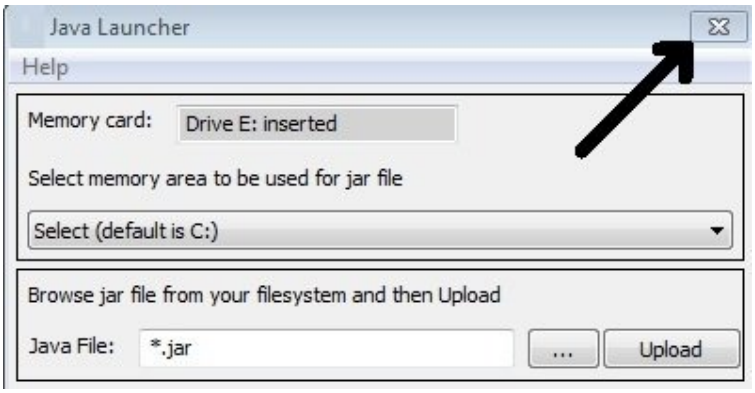

FIGURE 11. Exit button pointed with an arrow (Jari-Pekka Teurajärvi 2013).

6.3.3 Memory card information note

Since the user is able to select which destination memory is to be used in the simulated device for installing the MIDlet, it is valuable information to tell if other memory areas than the default one is available. Simulation tool has a support for several memory areas depending on the simulated device configuration. The simulated device may have for example an option for a memory card. In Figure 12 the information about the memory card option is pointed with an arrow.

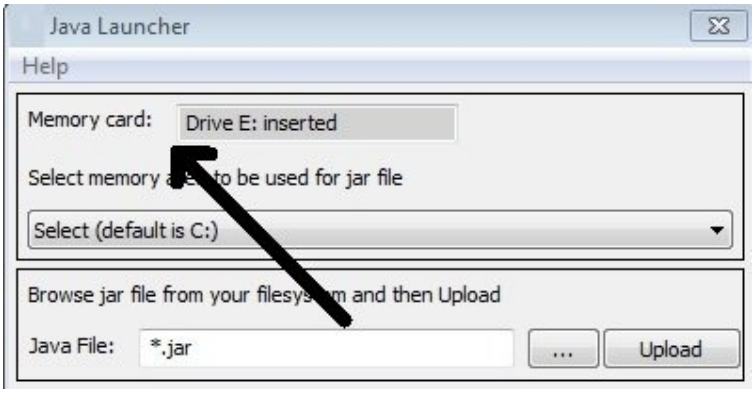

FIGURE 12. Memory card information pointed with an arrow (Jari-Pekka Teurajärvi 2013).

6.3.4 Select memory area to be user for JAR file

Java Launcher tool provides a possibility to select the destination memory area for the installable MIDlet in the simulated device. The possible memory areas available are dependent on the simulated device configuration. The default memory area is a drive C and that is also considered as the device internal memory area. Extendable memory areas are, for example memory cards. The drive letter is defined in the device configuration. With the selectable memory area, the real life methods of installing the MIDlets can be enabled for testing in a simulated environment, this also enables a quicker implementation cycle and finally a quicker software releasing time. Figure 13 shows the memory area selection as a drop-down list which allows the user to choose one value from a list.

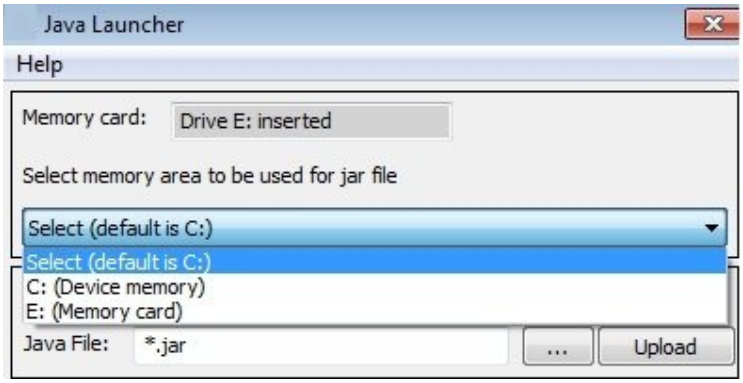

FIGURE 13. Select memory area drop-down list (Jari-Pekka Teurajärvi).

6.3.5 Browse button for browsing the JAR file from the file system

The Browse button enables the MIDlet browsing and selection from the host OS file system. To be able to install the MIDlet from the host OS file system to the simulated device file system, the MIDlet must be located. The user has to select only the JAR file since the JAD file will be copied automatically if valid. A valid JAD file is named equally to the JAR file and the content must be correct. The Browse button will display only the JAR files in the host OS file system and this will enable an easy selection of the proper MIDlet. In Figure 14 the Browse button is pointed with an arrow.

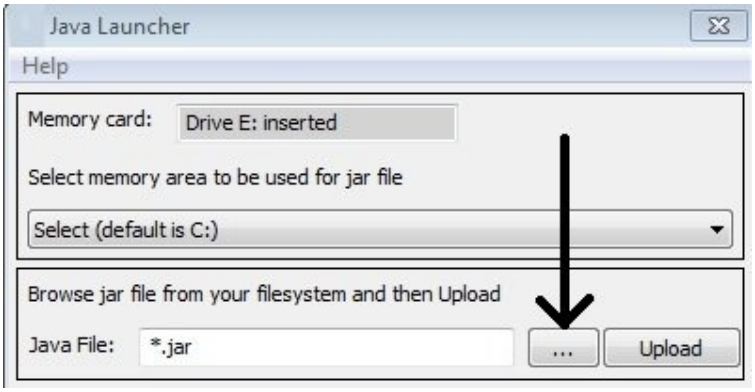

FIGURE 14. Browse button enables the installable MIDlet location and selection (Jari-Pekka Teurajärvi 2013).

6.3.6 Upload button for installing the MIDlet

The final phase of the installation is to upload the MIDlet from the host OS file system to the simulated device file system. The Upload button will initialize the copying of the user selected MIDlet files both JAR and JAD from the host OS file system to the simulation environment file system. In Figure 15 the Upload button is pointed with an arrow.

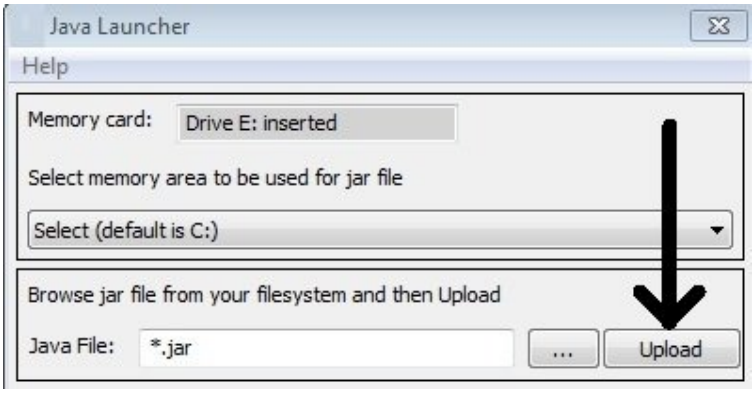

FIGURE 15. Upload button initializes the copying of the MIDlet file (Jari-Pekka Teurajärvi 2013).

6.4 MIDlet usage at the simulation environment

To be able to run the installed MIDlet in a simulation environment, the end user does not need to know the location of the MIDlet at the file system level but it is good to know how to start applications in that particular device. There can be several different memory areas in a device as well as in a simulated device where the MIDlet can be installed. The different memory areas can be configured using the device configuration of the simulated device. The configuration is not done by the end user but by the device manufacturer. There must be a default memory area that is usually the internal memory area of the device. The internal memory area can be divided into different sections depending on the usage. There can also be external or extendable memory areas like memory cards simulated into Simulation tool and they are visible in the simulated device, too. Figure 16 shows how to navigate installed applications like Java MIDlets.

FIGURE 16. An example of the navigation of the installed applications in a simulated device (Jari-Pekka Teurajärvi).

6.4.1 Using the installed MIDlet from the default memory area

It depends on the simulated device what is the directory hierarchy or the menu hierarchy where the installed application is located in the GUI. In Figure 17 you can see one example of how to navigate the installed MIDlet in a simulated device.

FIGURE 17. An example of the navigation of the installed applications in a default memory area (Jari-Pekka Teurajärvi 2013).

6.4.2 Using the installed MIDlet from the memory card memory area

There might be an external or extendable memory area in the simulated device configurations and that is visible also in the simulated device GUI. In Figure 18 you can see one example of how to navigate the installed MIDlet in a simulated device external memory area, in this case a memory card.

FIGURE 18. An example of the navigation of the installed applications in a memory card (Jari-Pekka Teurajärvi).

7 CONCLUSION

7.1 Summary

The end result of this Master's Thesis provides for programmers an easier way to copy Java MIDlets from the computer file system to the simulated device file system. Now Java MIDlet programmers do not have to know the actual path where to copy the MIDlet to the simulated device file system. Also, deleting the old version of the MIDlet is done on behalf of a programmer.

The actual implementation was done using the standard ANSI C coding language. Simulation tool provided good APIs for implementing the functionality and the GUI. The implementation process also revealed a missing API support of launching the MIDlet without a user interaction from the simulated device GUI. Before implementing an enhanced version of Java Launcher, the support for launching the MIDlets using the API must be implemented.

7.2 Consideration

As a future improvement of Java Launcher, it is mandatory to have the Simulation tool API support for launching the Java MIDlets using the API instead of using the GUI. This will enable the best possible user experience. One example is a Java MIDlet that is running a set of tests for certain purposes. When a programmer is implementing new test cases to the test set, Java Launcher would enable an easy way to install the new test set Java MIDlet and run the test set automatically right after the installation. With the current implementation, Java Launcher is only enabling the easy way of copying the test set Java MIDlet but a programmer has to launch the MIDlet manually from the simulated device GUI.

REFERENCES

1. Brian W. Kernighan, Dennis M. Ritchie, 2012. The C Programming Language. Date of retrieval 18.10.2013.
   http://zanasi.chem.unisa.it/download/C.pdf

2. Difference Between 2013. Difference Between Linux File System and Windows File System. Date of retrieval 18.10.2013.
   http://www.differencebetween.com/difference-between-linux-file-system-and-vs-windows-file-system/

3. Microsoft 2013. XAML Overview (WPF), .NET Framework 4.5, Microsoft Developer network. Date of retrieval 18.10.2013.
   http://msdn.microsoft.com/en-us/library/ms752059.aspx

4. Wikibooks 2013. C Programming/C Reference/stdio.h. Date of retrieval 18 October 2013.
   http://en.wikibooks.org/wiki/C_Programming/C_Reference/stdio.h

5. Wikipedia 2013. Connected Limited Device Configuration. Date of retrieval 18.10.2013.
   http://en.wikipedia.org/wiki/Connected_Limited_Device_Configuration

6. Wikipedia 2013. Java Platform, Micro Edition. Date of retrieval 18.10.2013.
   http://en.wikipedia.org/wiki/Java_ME

7. Wikipedia 2013. MIDlet. Date of retrieval 18.10.2013.
   http://en.wikipedia.org/wiki/MIDlet

8. Wikipedia 2013. Mobile Information Device Profile. Date of retrieval 18.10.2013.
   http://en.wikipedia.org/wiki/Mobile_Information_Device_Profile

9. Wikipedia 2013. Application programming interface. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Application_programming_interface

10. Wikipedia 2013. Interoperability. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Interoperability

11. Wikipedia 2013. Hardware abstraction. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Hardware_abstraction

12. Wikipedia 2013. Chipset. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Chipset

13. Wikipedia 2013. JAR (file format). Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/JAR_(file_format)

14. Wikipedia 2013. JAD (file format). Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/JAD_(file_format)

15. Wikipedia 2013. Drop-down list. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Drop-down_list

16. Wikipedia 2013. Compact disc. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Compact_disc

17. Wikipedia 2013. DVD. Date of retrieval 18.10.2013.
https://en.wikipedia.org/wiki/DVD

18. Wikipedia 2013. File Allocation Table. Date of retrieval 18.10.2013.
https://en.wikipedia.org/wiki/File_Allocation_Table

19. Wikipedia 2013. NTFS. Date of retrieval 18.10.2013.
https://en.wikipedia.org/wiki/NTFS

20. Wikipedia 2013. Extended file system. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Extended_file_system

21. Wikipedia 2013. Unix File System. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Unix_File_System

22. Wikipedia 2013. Path (computing). Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Path_(computing)

23. Wikipedia 2013. File system API. Date of data retrieval 18.10.2013.
http://en.wikipedia.org/wiki/File_system_API

24. Wikipedia 2013. C file input/output. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/C_file_input/output

25. Wikipedia 2013. Simulation. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Simulator

26. Wikipedia 2013. Emulator. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Emulator

27. Wikipedia 2013. Human-in-the-loop. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Human-in-the-Loop

28. Wikipedia 2013. Extensible Application Markup Language. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/Extensible_Application_Markup_Language

29. Wikipedia 2013. XML. Date of retrieval 18.10.2013.
http://en.wikipedia.org/wiki/XML

APPENDICES

JAVA LAUNCHER HEADER FILE APPENDIX 1

```
/*
Change History:

VERSION     : 1         03-Jun-2011     J-P Teurajärvi
REASON      : initial version of Java Launcher
DESCRIPTION : Initial version of Java Launcher for X

--------------------------------------------------------------------------------
*/
#ifndef NOSE_JAVA_LAUCHER_H
#define NOSE_JAVA_LAUCHER_H

#include "isa_simulation.hi"

#define SIMU_JAVA_LAUNCHER_PATH_MAX_LEN         255
#define SIMU_JAVA_LAUNCHER_DRIVE_DEFAULT        "Select (default is C:)"
#define SIMU_JAVA_LAUNCHER_DRIVE_C              "C: (Device memory)"
#define SIMU_JAVA_LAUNCHER_DRIVE_E              "E: (Memory card)"

#define SIMU_JAVA_LAUNCHER_UNDEFINED_DRIVE      0
#define SIMU_JAVA_LAUNCHER_DEFAULT_DRIVE        1
#define SIMU_JAVA_LAUNCHER_C_DRIVE              2
#define SIMU_JAVA_LAUNCHER_E_DRIVE              3

#endif /* NOSE_JAVA_LAUCHER_H */
```

```
/*

Change History:

VERSION    : 3        05.09.2011    J-P Teurajärvi
REASON     : Wrong usage of nose_channel_read(). File open harmonization between Windows and Linux. Layout
changes.
DESCRIPTION : nose_channel_read() is malfunctioning with Linux if used with nose_channel_callback. File open har-
monization between Windows and Linux. Layout changes.

VERSION    : 2        25.08.2011    J-P Teurajärvi
REASON     : problems with Linux compiler
DESCRIPTION : Building the source code with Linux compiler not successfull.

VERSION    : 1        03.06.2011    J-P Teurajärvi
REASON     : initial version
DESCRIPTION : Initial version of Java Launcher for NoSe
--------------------------------------------------------------------------------
*/

/* Global include files */
#include "global.h"
#include "product.h"
#include "type_def.h"
#include "filesys.h"

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "simu.h"
#include "ucs2_strlib.h"
#include "isa_simulation.hi"
#include "simu_java_launcher.h"

/* Local variables for file handling */
typedef unsigned short ucs2char;
static int selected_drive = SIMU_JAVA_LAUNCHER_DEFAULT_DRIVE;
static char jar_target_path_c[SIMU_JAVA_LAUNCHER_PATH_MAX_LEN];
static char jar_target_path_default[SIMU_JAVA_LAUNCHER_PATH_MAX_LEN];
static char jar_target_path_e[SIMU_JAVA_LAUNCHER_PATH_MAX_LEN];
static char user_jar_file_path[SIMU_JAVA_LAUNCHER_PATH_MAX_LEN];

/* FUNCTIONS */
static void simu_java_launcher_create_channels( void );
static void simu_java_launcher_failure_handler( NOSE_RETVAL value );
static void simu_java_launcher_shutdown( void );
static void simu_java_launcher_install_midlet( void );
static void simu_java_launcher_select_drive_handle( NOSE_HANDLE handle, void *data, size_t length, const void
*pUserData );
static void simu_java_launcher_select_source_handle( NOSE_HANDLE handle, void *data, size_t length, const void
*pUserData );
static void simu_java_launcher_upload_handle( const char *command );
static void simu_java_launcher_check_default_paths( void );
static NOSE_RETVAL simu_java_launcher_copy_paste_jar_files( const char *source, const char *target, char
**full_target );
static NOSE_RETVAL simu_java_launcher_copy_source_data( const char *source, long *file_size, char **whole_data );
static NOSE_RETVAL simu_java_launcher_open_file( const char *source, long *file_size, FILE **source_file );
static NOSE_RETVAL simu_java_launcher_read_data( long *file_size, FILE *source_file, char **whole_data );
static NOSE_RETVAL simu_java_launcher_fclose_file( FILE *source_file );
static NOSE_RETVAL simu_java_launcher_filesys_close_file( int32 fildes );
static NOSE_RETVAL simu_java_launcher_create_file_name( const char *source, const char *target, char **full_target,
ucs2char **ucs2_target );
static NOSE_RETVAL simu_java_launcher_paste_source_data( const char *source, const char *target, long file_size,
char *whole_data, char **full_target );
static NOSE_RETVAL simu_java_launcher_create_file( int32 *fildes, char **full_target, ucs2char *ucs2_target );
static NOSE_RETVAL simu_java_launcher_write_data( const char *target, int32 fildes, long file_size, char *whole_data
);

/* Handles for file processing */
extern NOSE_HANDLE hFileSys;
```

```
/* Handle for traces */
static NOSE_HANDLE hJavaLauncherTraceHandle;

/* Handles for NoSe Java Launcher window */
static NOSE_HANDLE hJavaLauncherSourceSelection;
static NOSE_HANDLE hJavaLauncherDriveSelection;

/*
================================================================================
*/
```

```
void simu_java_launcher_init( void )
{
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_init started." );

   /* Create Trace set for Java Launcher*/
   nose_trace_create( "Simulation", "Java Launcher", "Info", &hJavaLauncherTraceHandle );

   simu_shutdown_register( simu_java_launcher_shutdown );

   /* Create and attach channels for NoSe Java Launcher window */
   simu_java_launcher_create_channels( );

   /* Check the default paths for jar installation */
   simu_java_launcher_check_default_paths( );

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_init finished." );
}
/*
============================================================================
*/
```

```
static void simu_java_launcher_create_channels( void )
{
   NOSE_RETVAL retval = NOSE_OK;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_create_channels started." );

   nose_trace_printf( hJavaLauncherTraceHandle, "Create
SIMU_JAVA_LAUNCHER_DRIVE_SELECTION_CHANNEL" );
   retval = nose_channel_create( "SIMU_JAVA_LAUNCHER_DRIVE_SELECTION_CHANNEL",
&hJavaLauncherDriveSelection );
   if ( NOSE_OK != retval) { simu_java_launcher_failure_handler( retval ); }

   nose_trace_printf( hJavaLauncherTraceHandle, "Attach simu_java_launcher_select_drive_handle callback" );
   retval = nose_channel_callback_attach( hJavaLauncherDriveSelection, simu_java_launcher_select_drive_handle,
NULL );
   if ( NOSE_OK != retval) { simu_java_launcher_failure_handler( retval ); }

   nose_trace_printf( hJavaLauncherTraceHandle, "Create
SIMU_JAVA_LAUNCHER_SOURCE_SELECTION_CHANNEL" );
   retval = nose_channel_create( "SIMU_JAVA_LAUNCHER_SOURCE_SELECTION_CHANNEL",
&hJavaLauncherSourceSelection );
   if ( NOSE_OK != retval) { simu_java_launcher_failure_handler( retval ); }

   nose_trace_printf( hJavaLauncherTraceHandle, "Attach simu_java_launcher_select_source_handle callback" );
   retval = nose_channel_callback_attach( hJavaLauncherSourceSelection, simu_java_launcher_select_source_handle,
NULL );
   if ( NOSE_OK != retval) { simu_java_launcher_failure_handler( retval ); }

   EsimMsgHandlerRegisterNOS( "JAVA_LAUNCHER", simu_java_launcher_upload_handle );

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_create_channels finished." );
}
/*
================================================================================
*/
```

```
static void simu_java_launcher_shutdown( void )
{
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_shutdown started." );
   /* Local channels */
   nose_channel_callback_detach( hJavaLauncherDriveSelection );
   nose_channel_delete( &hJavaLauncherDriveSelection );
   nose_channel_callback_detach( hJavaLauncherSourceSelection );
   nose_channel_delete( &hJavaLauncherSourceSelection );
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_shutdown finished." );
}
/*
================================================================================
*/
```

```
static void simu_java_launcher_failure_handler( NOSE_RETVAL value )
{
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_failure_handler started." );
   switch ( value )
   {
      /* FileAPI specific values */
      case NOSE_NAME_TOO_LONG:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: Channel creation failed, NOSE_NAME_TOO_LONG"
);
         break;
      case NOSE_INVALID_NAME:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: Channel creation failed, NOSE_INVALID_NAME" );
         break;
      case NOSE_INVALID_HANDLE:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: Channel creation failed, NOSE_INVALID_HANDLE"
);
         break;
      case NOSE_INVALID_HANDLE_TYPE:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: Channel creation failed,
NOSE_INVALID_HANDLE_TYPE" );
         break;
      /* ChannelAPI specific values */
      case NOSE_INVALID_CALLBACK_FUNCTION:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: Channel creation failed,
NOSE_INVALID_CALLBACK_FUNCTION" );
         break;
      case NOSE_CALLBACK_ALREADY_ATTACHED:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: Channel creation failed,
NOSE_CALLBACK_ALREADY_ATTACHED" );
         break;
      /* FileAPI specific values */
      case NOSE_FILE_NOT_FOUND:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: File System failed, NOSE_FILE_NOT_FOUND" );
         break;
      case NOSE_FILE_ACCESS_DENIED:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: File System failed, NOSE_FILE_ACCESS_DENIED"
);
         break;
      case NOSE_FILE_EXISTS:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: File System failed, NOSE_FILE_EXISTS" );
         break;
      case NOSE_INVALID_OTYPE:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: File System failed, NOSE_INVALID_OTYPE" );
         break;
      case NOSE_INVALID_PMODE:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: File System failed, NOSE_INVALID_PMODE" );
         break;
      case NOSE_DISK_FULL:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: File System failed, NOSE_DISK_FULL" );
         break;
      case NOSE_FILE_HANDLES_EXCEEDED:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: File System failed,
NOSE_FILE_HANDLES_EXCEEDED" );
         break;
      /* Config */
      case NOSE_FAILED:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: Config reading failed, NOSE_FAILED" );
         break;
      /* Common */
      case NOSE_UNKNOWN_ERROR:
         nose_trace_printf( hJavaLauncherTraceHandle, "ERROR: Unknown error occured. Abort." );
         break;
      default:
         nose_trace_printf( hJavaLauncherTraceHandle, "NOSE STATUS: %d", value );
         break;
   }
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_failure_handler finished." );
}
/*============================================================================================*/
```

```
static void simu_java_launcher_install_midlet( void )
{
    char *target        = NULL;
    char *source        = NULL;
    int source_path_len = 0;
    int target_path_len = 0;
    NOSE_RETVAL retval  = NOSE_OK;
    char *full_target   = NULL;

    nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_install_midlet started." );
    /* set the selected target drive */
    if( SIMU_JAVA_LAUNCHER_DEFAULT_DRIVE == selected_drive )
    {
        nose_trace_printf( hJavaLauncherTraceHandle, "Default drive selected (C:)" );
        target = jar_target_path_default;
    }
    else if ( SIMU_JAVA_LAUNCHER_C_DRIVE == selected_drive )
    {
        nose_trace_printf( hJavaLauncherTraceHandle, "Drive selected (C:)" );
        target = jar_target_path_c;
    }
    else if( SIMU_JAVA_LAUNCHER_E_DRIVE == selected_drive )
    {
        nose_trace_printf( hJavaLauncherTraceHandle, "Memory card selected (E:)" );
        target = jar_target_path_e;
    }
    else
    {
        selected_drive = SIMU_JAVA_LAUNCHER_UNDEFINED_DRIVE;
        nose_trace_printf( hJavaLauncherTraceHandle, "ERROR with target drive selection" );
        simu_java_launcher_failure_handler( NOSE_UNKNOWN_ERROR );
        return;
    }
    nose_trace_printf( hJavaLauncherTraceHandle, "target path: %s", target );

    source_path_len = strlen( user_jar_file_path ) - 1;

    /* check the selected source path length validity */
    if ( ( NULL != user_jar_file_path ) && ( SIMU_JAVA_LAUNCHER_PATH_MAX_LEN >= source_path_len ) )
    {
        source = user_jar_file_path;
        nose_trace_printf( hJavaLauncherTraceHandle, "source path: %s", source );
    }
    else
    {
        nose_trace_printf( hJavaLauncherTraceHandle, "ERROR with source selection" );
        simu_java_launcher_failure_handler( NOSE_FILE_NOT_FOUND );
        return;
    }

    /* copy the selected *.jar file */
    retval = simu_java_launcher_copy_paste_jar_files( source, target, &full_target );

    /* a *.jad file linked to the previously copied *.jar file must be
     * also copied. We need to build the path to the *.jad file.
     * You are really reading this, email to me teurajarvi@hotmail.com
     * and I will buy you a cup of coffee.
     * The *.jad file should be at the very same source directory as
     * the *.jar file was.
     */
    if ( NOSE_OK == retval )
    {
        nose_trace_printf( hJavaLauncherTraceHandle, "change the source file suffix from jar to jad" );
        source[ source_path_len ] = 'd';

        /* copy the selected jad file */
        retval = simu_java_launcher_copy_paste_jar_files( source, target, &full_target );
    }

    /* Handle the possible failure */
    if ( NOSE_OK != retval )
    {
```

```
        simu_java_launcher_failure_handler( retval );
        /* If the *.jad file copying failed the *.jar file related is obsolete */
        if ( full_target != NULL )
        {
            target_path_len = strlen( full_target );
            full_target[ target_path_len ] = 'r';
            retval = nose_file_delete( hFileSys, full_target );
        }
        if ( NOSE_OK != retval )
        {
        simu_java_launcher_failure_handler( retval );
        }
    }
    /* Clears information from the JavaLauncher window */
    nose_channel_write( hJavaLauncherSourceSelection, " \0", strlen("\0") );

    /* full_target is obsolete now */
    if ( NULL != full_target )
    {
        free( full_target );
        full_target = NULL;
    }

    nose_trace_printf( hJavaLauncherTraceHandle, "Refresh the PhoneX File System." );
    retval = nose_filesys_refresh( hFileSys, NULL );
    if ( NOSE_OK != retval )
    {
        simu_java_launcher_failure_handler( retval );
    }

    nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_install_midlet finished." );
}
/*
================================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_copy_paste_jar_files( const char *source, const char *target, char **full_target )
{
   NOSE_RETVAL retval = NOSE_OK;
   long file_size     = 0;
   char *whole_data   = NULL;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_copy_paste_jar_files started." );

   retval = simu_java_launcher_copy_source_data( source, &file_size, &whole_data );

   if ( NOSE_OK == retval )
   {
      retval = simu_java_launcher_paste_source_data( source, target, file_size, whole_data, full_target );
   }

   /* whole_data is obsolete now */
   if ( NULL != whole_data )
   {
      free( whole_data );
      whole_data = NULL;
   }

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_copy_paste_jar_files finished." );
   return retval;
}
/*
================================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_copy_source_data( const char *source, long *file_size, char **whole_data )
{
   NOSE_RETVAL retval = NOSE_OK;
   FILE *source_file  = NULL;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_copy_source_data started." );

   /* Open the source file for reading */
   retval = simu_java_launcher_open_file( source, file_size, &source_file );

   /* Read the data from the source file */
   if ( NOSE_OK == retval )
   {
      retval = simu_java_launcher_read_data( file_size, source_file, whole_data );
   }
   else
   {
      return retval;
   }

   /* Close the source file */
   if ( NOSE_OK == retval )
   {
      retval = simu_java_launcher_fclose_file( source_file );
   }
   else
   {
      return retval;
   }

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_copy_source_data finished." );
   return retval;
}
/*
================================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_open_file( const char *source, long *file_size, FILE **source_file )
{
   NOSE_RETVAL retval  = NOSE_OK;
   int file_status     = 0;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_open_file started." );
   nose_trace_printf( hJavaLauncherTraceHandle, "path of the source file: %s", source );

   /* Open source file using the stdio library */
   *source_file = fopen( source, "rb" );
   if( NULL == *source_file )
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "Open source file FAILED." );
      return NOSE_FILE_NOT_FOUND;
   }

   /* Get the size of the file, first find the position of the EOF */
   file_status = fseek( *source_file, 0L, SEEK_END );
   if ( 0 != file_status )
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "Seeking the end of the source file FAILED." );
      simu_java_launcher_fclose_file( *source_file );
      return NOSE_UNKNOWN_ERROR;
   }

   /* Get the position */
   *file_size = ftell( *source_file );
   if ( -1 == *file_size )
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "Getting the size of the source file FAILED." );
      simu_java_launcher_fclose_file( *source_file );
      return NOSE_UNKNOWN_ERROR;
   }

   rewind( *source_file );

   nose_trace_printf( hJavaLauncherTraceHandle, "Size of the source file: %d bytes", *file_size );
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_open_file finished." );
   return retval;
}
/*
================================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_read_data( long *file_size, FILE *source_file, char **whole_data )
{
   NOSE_RETVAL retval = NOSE_OK;
   int32 data_size    = 0;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_read_data started." );
   /* Allocate memory for the file data */
   *whole_data = ( char* )calloc( *file_size, 1 );
   data_size = *file_size;
   if ( NULL == *whole_data || 0 == data_size )
   {
      if ( ( 0 == data_size ) && ( NULL != *whole_data ) )
      {
         free( *whole_data );
         *whole_data = NULL;
      }
      simu_java_launcher_fclose_file( source_file );
      nose_trace_printf( hJavaLauncherTraceHandle, "Data buffer memory allocation FAILED." );
      return NOSE_UNKNOWN_ERROR;
   }

   /* Read RAW data from file*/
   if ( NULL != source_file )
   {
      *file_size = fread( *whole_data, 1, data_size, source_file );
   }
   else
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "Read data from the file FAILED" );
      nose_trace_printf( hJavaLauncherTraceHandle, "Reason: no source file." );
      return NOSE_UNKNOWN_ERROR;
   }

   if ( *file_size != data_size )
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "Read data from the file FAILED" );
      nose_trace_printf( hJavaLauncherTraceHandle, "Reason: bytes read vs data available %d / %d", *file_size, da-
ta_size );
      simu_java_launcher_fclose_file( source_file );
      return NOSE_UNKNOWN_ERROR;
   }
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_read_data finished." );
   return retval;
}
/*
============================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_fclose_file( FILE *source_file )
{
   NOSE_RETVAL retval   = NOSE_OK;
   int return_value     = 0;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_fclose_file started." );
   if ( source_file )
   {
      return_value = fclose( source_file );
      source_file = NULL;
      if ( 0 != return_value )
      {
         nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_fclose_file FAILED." );
         return NOSE_UNKNOWN_ERROR;
      }
   }

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_fclose_file finished." );
   return retval;
}

/*
================================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_filesys_close_file( int32 fildes )
{
   NOSE_RETVAL retval   = NOSE_OK;
   int32 filesys_status = 0;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_filesys_close_file started." );
   if ( 0 <= fildes )
   {
      filesys_status = filesys_close( fildes );
      if ( FILESYS_ERROR == filesys_status )
      {
         nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_filesys_close_file FAILED, reason: %d.",
filesys_status );
         return NOSE_UNKNOWN_ERROR;
      }
   }
   else
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_filesys_close_file FAILED, reason: invalid
descriptor." );
      return NOSE_UNKNOWN_ERROR;
   }
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_filesys_close_file finished." );
   return retval;
}
/*
============================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_paste_source_data( const char *source, const char *target,
   long file_size, char *whole_data, char **full_target )
{
   NOSE_RETVAL retval    = NOSE_OK;
   int32 fildes          = FILESYS_ERROR;
   ucs2char *ucs2_target = NULL;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_paste_source_data started." );

   /* Create a filename for new file */
   retval = simu_java_launcher_create_file_name( source, target, full_target, &ucs2_target );

   /* Create a new file to the target destination */
   if ( NOSE_OK == retval )
   {
      retval = simu_java_launcher_create_file( &fildes, full_target, ucs2_target );
   }
   else
   {
      return retval;
   }

   /* Write the data from the buffer to the new target file */
   if ( NOSE_OK == retval )
   {
      retval = simu_java_launcher_write_data( target, fildes, file_size, whole_data );
   }
   else
   {
      return retval;
   }

   /* Close the file just created */
   if ( NOSE_OK == retval )
   {
      retval = simu_java_launcher_filesys_close_file( fildes );
   }

   /* ucs2_target is obsolete now */
   if ( NULL != ucs2_target )
   {
      free( ucs2_target );
      ucs2_target = NULL;
   }

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_paste_source_data finished." );
   return retval;
}
/*
================================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_create_file_name( const char *source, const char *target,
    char **full_target, ucs2char **ucs2_target )
{
    NOSE_RETVAL retval = NOSE_OK;
    char *filename     = NULL;

    nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_create_file_name started." );
    nose_trace_printf( hJavaLauncherTraceHandle, "Destination path: %s", target );

    /* Create a filename for jar/jad file */
    filename = strrchr( source, SIMU_PATH_SEP ); /* find the last separator from the path */
    filename++; /* skip the separator */
    nose_trace_printf( hJavaLauncherTraceHandle, "filename: %s", filename );

    /* Merge the directory path and filename. User must free the memory allocated by JOIN2. */
    *full_target = JOIN2( target, filename);
    nose_trace_printf( hJavaLauncherTraceHandle, "Full target path: %s", *full_target );

    *ucs2_target = calloc( 2 * strlen( *full_target ) + 2, sizeof( ucs2char ) );
    (void)ucs2_strcpy_ascii_to_ucs2( *ucs2_target, *full_target );

    /* Error check for conversions */
    if ( NULL == *ucs2_target )
    {
        nose_trace_printf( hJavaLauncherTraceHandle, "ucs2 conversion failure." );
        /* Free memory to avoid memory leaks */
        if ( NULL != *full_target )
        {
            free( *full_target );
            *full_target = NULL;
        }
        return NOSE_UNKNOWN_ERROR;
    }
    nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_create_file_name finished." );
    return retval;
}
/*
================================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_create_file( int32 *fildes, char **full_target, ucs2char *ucs2_target )
{
   NOSE_RETVAL retval = NOSE_OK;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_create_file started." );

   *fildes = filesys_open( ( const char16* )ucs2_target, FILESYS_O_CREAT | FILESYS_O_RDWR |
FILESYS_O_EXCL, 0 );
   nose_trace_printf( hJavaLauncherTraceHandle, "File descriptor: %d", *fildes );
   if ( 0 > *fildes )
   {
      /* There might be an old version of the file existing */
      nose_trace_printf( hJavaLauncherTraceHandle, "Old version of the file existing -> starting to delete." );
      retval = nose_file_delete( hFileSys, *full_target );
      if ( NOSE_OK != retval )
      {
         nose_trace_printf( hJavaLauncherTraceHandle, "Delete FAILED." );
         return retval;
      }
      else
      {
         /* Delete of the old file version done. Create a new file. */
         nose_trace_printf( hJavaLauncherTraceHandle, "Delete finished. Try to create a file again." );
         *fildes = filesys_open( ( const char16* )ucs2_target, FILESYS_O_CREAT | FILESYS_O_RDWR |
FILESYS_O_EXCL, 0 );
         nose_trace_printf( hJavaLauncherTraceHandle, "File descriptor: %d", *fildes );
      }
   }

   if ( 0 <= *fildes )
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "Create destination file finished successfully." );
   }
   else
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "Unexpected Filesys failure occured. Abort." );
      return NOSE_UNKNOWN_ERROR;
   }
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_create_file finished." );
   return retval;
}
/*
================================================================================
*/
```

```
NOSE_RETVAL simu_java_launcher_write_data( const char *target, int32 fildes, long file_size, char* whole_data )
{
   NOSE_RETVAL retval  = NOSE_OK;
   int32 bytes_written = 0;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_write_data started." );

   /* Write the data from the buffer to the target file. */
   bytes_written = filesys_write( fildes, whole_data, file_size );
   nose_trace_printf( hJavaLauncherTraceHandle, "%d bytes written", bytes_written );
   if (  0 == bytes_written || FILESYS_ERROR == bytes_written )
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "File writing FAILED." );
      return NOSE_UNKNOWN_ERROR;
   }
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_write_data finished." );
   return retval;
}
/*
================================================================================
*/
```

JAVA LAUNCHER SELECT DRIVE HANDLE APPENDIX 18

```
static void simu_java_launcher_select_drive_handle( NOSE_HANDLE handle, void *data, size_t length, const void
*pUserData )
{
   NOSE_RETVAL retval      = NOSE_OK;
   uint32 total_sectors    = 0;
   uint32 free_sectors     = 0;
   uint32 sector_size      = 0;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_select_drive_handle started." );

   /* Determine which value was selected */
   if( 0 == strcmp( ( char * ) data, SIMU_JAVA_LAUNCHER_DRIVE_DEFAULT ) )
   {
      selected_drive = SIMU_JAVA_LAUNCHER_DEFAULT_DRIVE;
      nose_trace_printf( hJavaLauncherTraceHandle, "Default drive selected (C:)" );
      retval = nose_file_drivestat( hFileSys, 'C', &total_sectors, &free_sectors, &sector_size );
      nose_trace_printf( hJavaLauncherTraceHandle, "path: %s", jar_target_path_default );
   }
   else if( 0 == strcmp( ( char * ) data, SIMU_JAVA_LAUNCHER_DRIVE_C ) )
   {
      selected_drive = SIMU_JAVA_LAUNCHER_C_DRIVE;
      nose_trace_printf( hJavaLauncherTraceHandle, "C drive selected (C:)" );
      retval = nose_file_drivestat( hFileSys, 'C', &total_sectors, &free_sectors, &sector_size );
      nose_trace_printf( hJavaLauncherTraceHandle, "path: %s", jar_target_path_c );
   }
   else if( 0 == strcmp( ( char * ) data, SIMU_JAVA_LAUNCHER_DRIVE_E ) )
   {
      selected_drive = SIMU_JAVA_LAUNCHER_E_DRIVE;
      nose_trace_printf( hJavaLauncherTraceHandle, "Memory card selected (E:)" );
      retval = nose_file_drivestat( hFileSys, 'E', &total_sectors, &free_sectors, &sector_size );
      nose_trace_printf( hJavaLauncherTraceHandle, "path: %s", jar_target_path_e );
   }
   else
   {
      selected_drive = SIMU_JAVA_LAUNCHER_UNDEFINED_DRIVE;
      nose_trace_printf( hJavaLauncherTraceHandle, "ERROR with drive selection" );
      retval = NOSE_UNKNOWN_ERROR;
   }

   /* Handle the possible drive failure */
   if ( NOSE_OK != retval )
   {
      simu_java_launcher_failure_handler( retval );
   }
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_select_drive_handle finished." );
}

/*
================================================================================
*/
```

```
static void simu_java_launcher_upload_handle( const char *command )
{
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_upload_handle started" );
   if( strcmp( command, "UPLOAD" ) == 0 )
   {
      simu_java_launcher_install_midlet( );
   }
   else
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_upload_handle FAILED." );
      simu_java_launcher_failure_handler( NOSE_UNKNOWN_ERROR );
   }
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_upload_handle finished" );
}
/*
================================================================================
*/
```

```
static void simu_java_launcher_select_source_handle( NOSE_HANDLE handle, void *data, size_t length, const void
*pUserData )
{
   char *fullPath       = NULL;
   int i                = 0;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_select_source_handle started." );
   /* Erase the old file path if existing */
   for ( i = 0; i < SIMU_JAVA_LAUNCHER_PATH_MAX_LEN; i++ )
   {
      user_jar_file_path[ i ] = 0;
   }

   /* Read from SIMU_JAVA_LAUNCHER_SOURCE_SELECTION_CHANNEL -channel the name of the file entered by
the user. */
   if ( SIMU_JAVA_LAUNCHER_PATH_MAX_LEN <= length )
   {
      simu_verify_fatal( 0, "SIMULATION ERROR: Too long File/path in Java Launcher window");
      nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_select_source_handle FAILED." );
   }
   else
   {
      strncpy( user_jar_file_path, data, length );
      user_jar_file_path[ length ] = 0; /* Force NUL to the end */
   }

   nose_trace_printf( hJavaLauncherTraceHandle, "user file: %s", user_jar_file_path );

   /* Convert to full path - BAD CODE WARNING : IF there's a separator, it's assumed as full path */
#if ( SIMULATION_ENVIRONMENT == G_SIMULATION_ENVIRONMENT_WINDOWS )
   fullPath = strchr( user_jar_file_path, '\\' );
#else
   fullPath = strchr( user_jar_file_path, '/' );
#endif
   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_select_source_handle finished." );
}
/*
================================================================================
*/
```

```
static void simu_java_launcher_check_default_paths( void )
{
   NOSE_RETVAL retval = NOSE_OK;

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_check_default_paths started." );

   /* Get target path for the C drive */
   retval = nose_config_read_string("JAVA_LAUNCHER/PATH_C", "PATH", jar_target_path_c,
SIMU_JAVA_LAUNCHER_PATH_MAX_LEN );
   if ( NOSE_OK != retval )
   {
      /* Set path for the C drive */
      simu_java_launcher_failure_handler( retval );
      strcpy( jar_target_path_c, "C:\\predefjava\\predefcollections\\" );
      nose_config_write_string( "JAVA_LAUNCHER/PATH_C", "PATH", jar_target_path_c );
      nose_trace_printf( hJavaLauncherTraceHandle, "C drive target path: %s", jar_target_path_c );
   }
   else
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "C drive target path: %s", jar_target_path_c );
   }

   /* Get target path for the default drive (C) */
   retval = nose_config_read_string( "JAVA_LAUNCHER/PATH_DEFAULT", "PATH", jar_target_path_default,
SIMU_JAVA_LAUNCHER_PATH_MAX_LEN );
   if ( NOSE_OK != retval )
   {
      /* Set target path for the *.jar files to be installed into PhoneX C default drive */
      simu_java_launcher_failure_handler( retval );
      strcpy( jar_target_path_default, "C:\\predefjava\\predefcollections\\" );
      nose_config_write_string( "JAVA_LAUNCHER/PATH_DEFAULT", "PATH", jar_target_path_default );
      nose_trace_printf( hJavaLauncherTraceHandle, "Default drive target path: %s", jar_target_path_default );
   }
   else
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "Default drive target path: %s", jar_target_path_default );
   }

   /* Get target path for the E drive */
   retval = nose_config_read_string( "JAVA_LAUNCHER/PATH_E", "PATH", jar_target_path_e,
SIMU_JAVA_LAUNCHER_PATH_MAX_LEN );
   if ( NOSE_OK != retval )
   {
      /* Set target path for the *.jar files to be installed into PhoneX E drive */
      simu_java_launcher_failure_handler( retval );
      strcpy( jar_target_path_e, "E:\\" );
      nose_config_write_string( "JAVA_LAUNCHER/PATH_E", "PATH", jar_target_path_e );
      nose_trace_printf( hJavaLauncherTraceHandle, "E drive target path: %s", jar_target_path_e );
   }
   else
   {
      nose_trace_printf( hJavaLauncherTraceHandle, "E drive target path: %s", jar_target_path_e );
   }

   nose_trace_printf( hJavaLauncherTraceHandle, "simu_java_launcher_check_default_paths finished." );
}
/*
================================================================================
*/
```